Automated Flight Dynamics Product Generation for the EOS AM-1 Spacecraft

Carla Matusow
Robert Wiegand
National Aeronautics and Space Administration
Goddard Space Flight Center
Code 583
Greenbelt, MD 20771
USA
Carla.Matusow@gsfc.nasa.gov
Robert.Wiegand@gsfc.nasa.gov

Abstract

Because of the complexity of the AM-1 spacecraft, the mission operations center requires more than 80 distinct flight dynamics products (reports). To create these products, the flight operations team will use a variety of modified commercial and National Aeronautics and Space Administration (NASA) developed flight dynamics software applications. Unfortunately, this means routine product generation requires flight dynamics expertise, requires skills in using each software application, takes several hours of operator interaction, and has many opportunities for user errors.

To address these issues, we (the flight dynamics team) developed automation software, called AutoProducts, which provides all the necessary coordination and communication among the various flight dynamics applications. AutoProducts autonomously retrieves files; sequences, initializes, and executes applications; and delivers the final products to the appropriate customers. This eliminates the need for flight dynamics expertise and knowledge of each application for routine product generation. Also, it virtually eliminates the potential for error and routine product generation needs no human-computer interaction.

Although AutoProducts required a significant effort to develop because of the complexity of the interfaces involved, its use will provide significant time and cost savings through reduced operator time and maximum product reliability. User satisfaction is significantly improved with AutoProducts and flight dynamics experts have more time to perform valuable analysis work. In addition, most of the AutoProducts code can be easily reused for future missions.

Key words: automation, autonomous operations, commercial-off-the-shelf, flight dynamics.

Introduction

As part of National Aeronautics and Space Administration's (NASA's) Earth Science Enterprise, the Earth Observing System (EOS) AM-1 spacecraft is designed to monitor long-term global environmental changes. Using the spacecraft flight operations system, EOS Mission Operations System (EMOS), the flight operators send commands and receive telemetry from AM-1 (Figure 1). The AM-1 Flight Dynamics System (FDS) provides important information to EMOS, which is used to create spacecraft commands and plan mission events. The flight dynamics information is packaged into more than 80 distinct products (reports). Specifically, the operators use FDS to:

- monitor spacecraft attitude and orbit in real-time
- provide spacecraft commanding information
- support anomaly resolution for spacecraft navigation, maneuver, and attitude systems
- perform spacecraft maneuver planning
- analyze spacecraft on-board orbit computations
- predict potential spacecraft communication times
- provide spacecraft science instrument planning aids

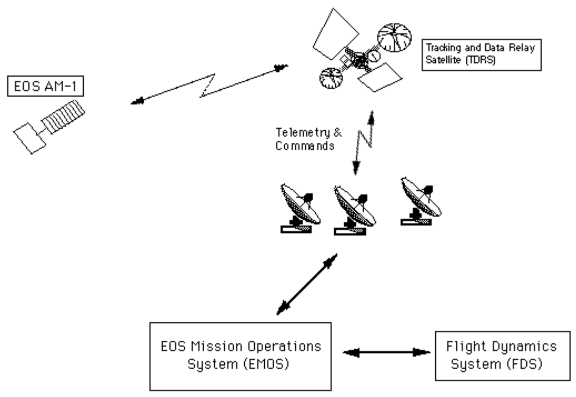

Tracking and Data Relay Satellite (TDRS)
EOS AM-1
Telemetry & Commands
EOS Mission Operations System (EMOS)
Flight Dynamics System (FDS)

Figure 1: AM-1 Flight Operations Ground System Components

Because the mission-required flight dynamics information is so diverse, FDS consists of several integrated commercial and custom software applications. Without any automation software, FDS was extremely cumbersome, requiring flight dynamics expertise and knowledge of several software packages and hardware platforms. Routine product generation was a highly interactive process that took hours to complete and had the potential for many user errors.

Problem description

In an effort to reduce software development costs and shorten development time, the flight dynamics organization at NASA regularly evaluates the use of commercial software packages. Therefore, FDS incorporates Satellite Tool Kit (STK) by Analytical Graphics, Inc., FreeFlyer by AI Solutions, Inc., and MATLAB by The Math Works, Inc to meet the AM-1 requirements (Figure 2). However, all the commercial software needed customization. Even after customizing the commercial software, we still needed to write some unique software to meet requirements specific to AM-1.

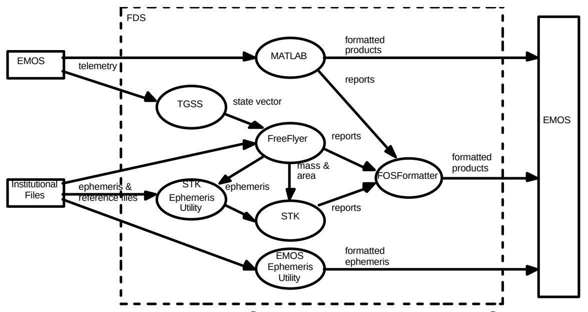


Figure 2: FDS Architecture Diagram Without AutoProducts

The STK package allows users to compile custom software modules with the commercially available STK modules to expand its functionality. We wrote several custom modules (in C) to produce acquisition data, lunar beta angles, lunar eclipse times, solar terminator, solar eclipse, solar mid day, spacecraft shadow times, and minimum/maximum latitude times – AM-1 specific functionality that wasn't available in the commercial package. We use built-in functionality to generate fields of view, viewing times, orbital events, solar and lunar azimuth and elevation angles, and altitude reports. STK uses predicted ephemeris files generated by FreeFlyer to produce its reports.

The FreeFlyer package provides an object oriented scripting language to allow users to perform tasks. We wrote several complex FreeFlyer scripts to produce maneuver planning products. We use relatively simple FreeFlyer scripts for predicted ephemeris, state vectors, and geometrical event reports (e.g., Brouwer-Lyddane elements, solar beta angles, and local sun time). FreeFlyer uses institutional reference files and a state vector from the spacecraft telemetry to produce its reports.

The MATLAB package requires users to write scripts using MATLAB library routines to perform tasks. We wrote several custom MATLAB modules to create a variety of spacecraft attitude related products. Our customized MATLAB modules use telemetry files to create the attitude data and produce several products in the EMOS format.

AM-1 has on-board navigation software, called the Tracking and Data Relay Satellite (TDRS) On-board Navigation System (TONS). A copy of the TONS flight software, implemented on a Sun Microsystems workstation, as well as some supporting software, is used to analyze spacecraft on-board orbit computations and performance. This integrated set of software is called the TONS Ground Support System (TGSS). TGSS uses telemetry files to validate orbit vectors from the spacecraft. FreeFlyer needs this information to generate its reports.

We modified existing NASA-developed ephemeris file reformatting software (Ephemeris Utilities) to produce the specific formats required by STK and EMOS.

Each product has precise data format requirements. Because FreeFlyer and STK cannot meet the formats required by EMOS, we wrote a custom application using Perl, called FOSFormatter, to reformat reports into the formats EMOS requires. Nearly all products must be reformatted by FOSFormatter before being delivered to the customers.

Every day, the flight operations team is required to generate 30 different flight dynamics products. Under special circumstances, even more products must be generated. From the flight operators' perspective, generating the daily products is a time-consuming task. The instructions to generate these products take 18 pages! Integrating several software applications into the FDS raises several operational concerns:

- Routine product generation requires knowledge of multiple applications executing on different hardware platforms. Generating daily products requires knowledge of UNIX, Windows NT, TGSS, STK, FreeFlyer, MATLAB, FOSFormatter, and the AM-1 Ephemeris Utilities.
- Generating products is a highly interactive process requiring a user to interact with each application multiple times to generate each product.

- Routine product generation requires several hours to complete. Although each daily product can be generated in 5-40 minutes, the entire process takes 6-8 hours because each product must be generated individually and serially – the first product must be completed before the next product can be started.
- User interaction with each application introduces the potential for errors, since users are required to manually enter filenames and input parameters as well as sequence and execute applications. Even with 18 pages of detailed instructions for daily product generation, operators commonly make several mistakes.
- Generating products requires some level of flight dynamics expertise to determine appropriate inputs and sequencing. Operators need to understand the system fairly well in order for product generation to make sense and prevent critical mistakes.

During FDS development these issues became apparent when developers and flight dynamics analysts began generating sample products for testing purposes. It quickly became clear that using several different software applications was unreasonable and some sort of automation was necessary to make the system easier to use.

Solution Description

To address these issues, we developed a software application called AutoProducts. The purpose of AutoProducts is to capture operations procedures. AutoProducts performs routine product generation without human-computer interaction. In addition, it acts as a single graphical user interface for all the flight dynamics software applications to allow users to generate unique groups of non-routine products easily.

AutoProducts is executed from a single hardware platform and provides all necessary coordination and communication among the various flight dynamics software applications. AutoProducts autonomously retrieves necessary files, sequences and executes applications (on the same platform and on other hardware platforms) with correct input parameters, and delivers the final flight dynamics products to the appropriate customers. Although AutoProducts will normally generate pre-programmed sets of routine products, its graphical user interface allows for configuration of customized and one-of-a-kind products. Additionally, AutoProducts has been designed as a mission-independent tool, and can be reconfigured to support other missions or incorporate new flight dynamics software applications. AutoProducts is capable of generating the appropriate products automatically at pre-determined time intervals for the life of the mission.

AutoProducts Framework

AutoProducts is an extensible framework. The key to understanding AutoProducts is the concept of Actions (Figure 3). An Action holds data that parameterizes a task that AutoProducts carrys out. Support for various kinds of Actions are implemented in modules that are loaded into AutoProducts at run time. For a particular kind of Action, the Action Execution module verifies and performs the Action. The Action Editor module displays and allows the user to modify the Action's data.

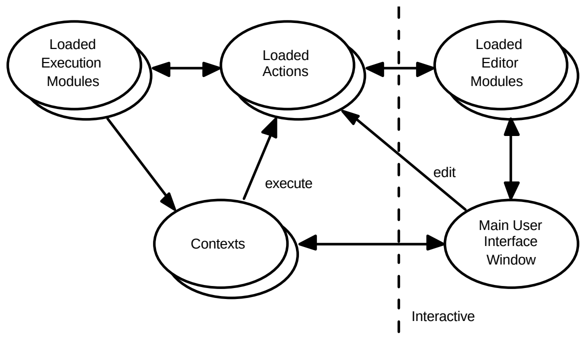


Figure 3: AutoProducts Framework

Actions fit nicely into the object oriented programming model. An Action has a Type, which defines how it implements the Action interface. Every Action understands the *validate*, *execute*, *dependencies*, and *edit* messages. The *validate* message requests a consistency check on the Action's data. The *execute* message carries out the Action with its current data (configuration). The *dependencies* message requests a list of the Actions that this Action is dependent upon. When AutoProducts runs interactively, the *edit* message presents an Action Editor to the user. Through the Action Editor, users customize the Action's data.

An example of a kind of Action is a List Action which groups and orders other Actions. Its data consists of a sequence of Actions and a flag for each indicating whether or not it is currently enabled. When a List is executed, it executes each of its enabled elements. Users enable or disable Actions within the List and add or remove Actions to the List through the List Editor.

An Action executes within a Context. The Context preserves state across Action executions. Actions communicate with each other by manipulating the

Context. The Action may have effects outside of AutoProducts (e.g. running another application), but coordinating these effects is handled through the Context. The Context maintains a stack of currently executing Actions. Contexts allow AutoProducts to perform Actions concurrently. The user interface allows creating a new Context and selecting the Context in which Actions will execute.

Actions and Types are maintained in a Registry (Figure 4). Loading an Action Execution module defines a new Type. The Type maps the *validate*, *execute*, *dependencies*, and *edit* messages to functions in the loaded modules. Actions (instances of Types) are loaded from files. The Registry verifies the Actions it loads with the *validate* and *dependencies* messages. Actions are also verified when the user attempts to commit an update (made with an Action Editor) to the Registry.

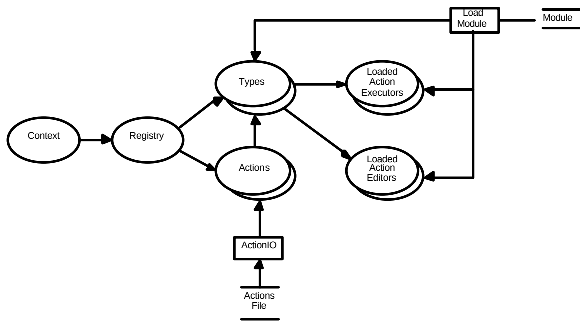


Figure 4: AutoProducts Actions and Types

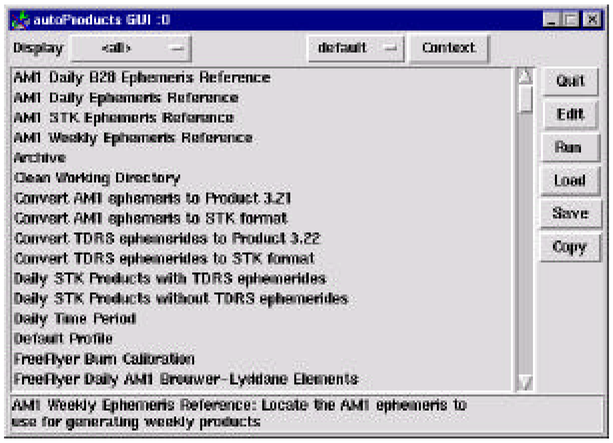
Each Context has its own Registry of Actions and Types. This is necessary since Actions may change in one Context (due to user interactions) while another Context is active. For Actions to safely execute concurrently, the Action execution module must maintain state in the Context and use the Context-unique identifier in external interactions. However, AutoProducts cannot guarantee that the application it is interacting with can perform the requested tasks in parallel.

Some Types of Actions have general application. A List Action groups a set of Actions in a particular sequence. A Test Action executes another Action and compares the results to the expected results. An Eval Action provides a hook to the implementation language.

The following are other key parts of the framework. The Messaging module implements the protocol for sending and receiving messages within AutoProducts. The Context service creates and commands existing Contexts (e.g. tells a Context to execute an Action). The Scheduling service schedules Actions for execution in a particular Context at a later time. The Reporting module collects and distributes status, warning, and error messages.

The AutoProducts framework allows modules to be plugged in when necessary. Adding support for a new Type of Action is simply a matter of implementing the Action interface. This might extend AutoProducts capabilities by interfacing with another software application. Implementing an Action Editor would support interactive operation. Different Action Editors, with more or less flexibility, can be plugged in for different users. Modules that do not directly support Actions can be loaded as well, for example, the module that provides support for AM-1 naming conventions.

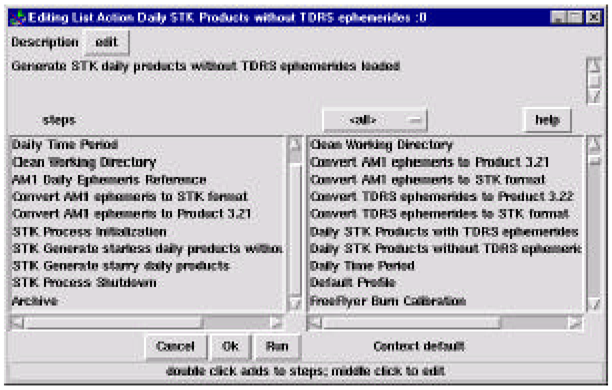
When run interactively, AutoProducts presents a Perl/Tk based user interface. Figure 5 shows the main user interface window that presents the list of available Actions. When the user requests to edit an Action, an Editor for that Action is opened (Figure 6). That Editor may open other windows (Figure 7). All of the Editors in the user interface populate a window having a common shell for consistency.



AM1 Daily B28 Ephemeris Reference
AM1 Daily Ephemeris Reference
AM1 STK Ephemeris Reference
AM1 Weekly Ephemeris Reference
Archive
Clean Working Directory
Convert AM1 ephemeris to Product 3.21
Convert AM1 ephemeris to STK format
Convert TDRS ephemerides to Product 3.22
Convert TDRS ephemerides to STK format
Daily STK Products with TDRS ephemerides
Daily STK Products without TDRS ephemerides
Daily Time Period
Default Profile
FreeFlyer Burn Calibration
FreeFlyer Daily AM1 Brouwer-Lyddane Elements

AM1 Weekly Ephemeris Reference: Locate the AM1 ephemeris to use for generating weekly products

Figure 5: AutoProducts Main User Interface Window With AM-1 Actions Loaded



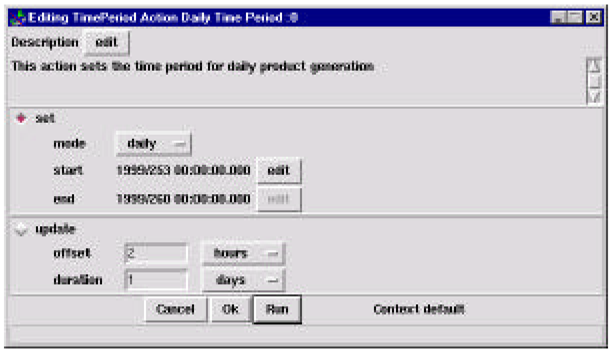
Editing List Action Daily STK Products without TDRS ephemerides :0

Generate STK daily products without TDRS ephemerides loaded

```
Daily Time Period                              Clean Working Directory
Clean Working Directory                        Convert AM1 ephemeris to Product 3.21
AM1 Daily Ephemeris Reference                  Convert AM1 ephemeris to STK format
Convert AM1 ephemeris to STK format            Convert TDRS ephemerides to Product 3.22
Convert AM1 ephemeris to Product 3.21          Convert TDRS ephemerides to STK format
STK Process Initialization                     Daily STK Products with TDRS ephemerides
STK Generate starless daily products withou    Daily STK Products without TDRS ephemeric
STK Generate starry daily products             Daily Time Period
STK Process Shutdown                           Default Profile
Archive                                        FreeFlyer Burn Calibration
```

double click adds to steps; middle click to edit

Figure 6: Sample List Action Editor Window



Editing TimePeriod Action Daily Time Period :0

This action sets the time period for daily product generation

```
◆ set
    mode       daily
    start      1999/253 00:00:00.000   edit
    end        1999/260 00:00:00.000   edit
◇ update
    offset     2     hours
    duration   1     days
```

Figure 7: Sample Action Editor Window

AM-1 Specific Implementation

To support the AM-1 mission, we developed a number of Action modules. We expect that many of these modules will be useful in future AutoProducts applications.

Supporting an external tool is dependent on the interface it provides. We were fortunate to have source code for many of the tools we need to support, simple interfaces to others (e.g., many UNIX utilities), and scripting languages and/or good vendor support for others. Trying to get events into an application that only provided a graphical interface would be a problem.

For AM-1, AutoProducts needs to interface with several software applications. We created Actions to work with each application: STK, FreeFlyer, MATLAB, FOSFormatter, and the Ephemeris Utilities. STK provides an external interface with its Connect module. AutoProducts communicates with STK/Connect via sockets. FreeFlyer provides its own scripting language. We annotated the AM-1 FreeFlyer scripts with hooks to allow automation. An operating system service on the FreeFlyer platform receives commands from AutoProducts to set up and execute FreeFlyer scripts. MATLAB also has a scripting language. We have a family of Actions that prepare MATLAB scripts for execution. Since FOSFormatter was written in Perl, its modules are easily loaded into AutoProducts at run-time. The Ephemeris Utilities are FORTRAN executables with Tcl/Tk user interfaces. AutoProducts communicates with the Ephemeris Utilities via standard UNIX input and output streams.

In addition to the application interfaces, we need infrastructure Actions to coordinate the product generation process. An FTP Action retrieves or delivers files using File Transfer Protocol (FTP). A Reference Action moves or renames a file. An Archive Action accesses or maintains the product archive. An Ephref Action selects and validates an ephemeris file for a satellite. A TimePeriod Action selects the timespan for products. A TDRSSlot Action selects the operational TDRS to use for product generation.

Implementation Notes

We implemented AutoProducts under HP-UX 10.20 using Perl 5. We chose HP-UX because it supports the largest subset of the software applications AutoProducts needs to coordinate. Perl 5 has strong module support and makes it easy to dynamically load modules. Also, Perl 5 has an extensive library and a good interface with the operating system.

Contexts are currently mapped onto processes. We intend to have the option of mapping Contexts onto threads when the Perl support for threads stabilizes. Threads offer two advantages. First, threads will be faster: there is some synchronization cost, but moving an object between queues in memory is much faster than serializing the object and transferring it across a pipe or socket. Second, threads will reduce resource usage since, for example, loaded modules can be shared.

It is possible to implement a non-Perl/Tk based set of editors for AutoProducts to load. For example, we could build a family of editors based on *curses* or editors that provide a World Wide Web (WWW) interface to AutoProducts.

Example

One tool that AutoProducts interfaces with is the Magnetic Field Prediction Utility (MFPU). Upon request, FDS provides EMOS with Three-Axis Magnetometer (TAM) fault detection isolation and recovery (FDIR) predict tables. These tables (created by MFPU) are required for on-board fault detection during attitude maneuvers. MFPU is implemented as a MATLAB application with its own user interface. The procedure to produce and deliver this table requires several pages of instructions. However, all the information necessary to perform the procedure can be derived from the product start time.

Within AutoProducts, we create a List Action called Produce TAM FDIR Predict Table. This List Action contains several Actions. First the Product Time Period Action specifies the start time of the product. This is the only place the user needs to specify information. Then the AM-1 Period Ephemeris Reference Action locates the AM 1 ephemeris file that covers that time period. Next, the MATLAB MFPU Action:

- derives the MFPU parameters from its data and the Context;
- builds a file containing statements to initialize and invoke the function that generates the product; and
- executes that file.

The FTP Products to EMOS Action delivers the product to EMOS. Lastly, the Archive Products Action archives the product. Using AutoProducts, the instructions for generating the TAM FDIR Predict Table are reduced to a few mouse clicks and selecting a start time.

Conclusion

AutoProducts greatly reduces many of the concerns associated with the flight dynamics product generation:

- Users can be trained to use a single application and graphical user interface to generate products. Groups of non-routine products can be generated with a few minutes of human-computer interaction using a single user interface. Users require knowledge of HP-UX and AutoProducts only.
- Routine product generation is performed autonomously. No interaction is necessary, allowing the operator to do other tasks during product generation.
- Routine product generation processing time has been significantly reduced. Daily product generation takes less than an hour and is solely dependent on the flight dynamics software applications' processing time.
- Since input parameters, filenames, and application sequences are preprogrammed for routine product generation, user interaction error is eliminated.
- No flight dynamics expertise is required for routine product generation. Autonomous operation reduces the level of expertise needed for flight operations.

Over the life of the AM-1 mission we expect AutoProducts to save at least 6 hours of operator time every day.

While EMOS is still being tested, flight dynamics analysts are frequently asked to provide sample products. With AutoProducts, generating sample products requires less lead-time and has a significantly smaller impact on the analysts' other work. They used to have to devote days to generating sample products – hours of human-computer interaction. With AutoProducts, only a few minutes of human-computer interaction is needed. Saving one analyst 24 hours of work to prepare sample products for each of 20 remaining tests before AM-1 launches, means an overall savings of at least 480 hours of analysts' time by using AutoProducts.

In addition, training users to use AutoProducts instead of all the flight dynamics software applications is easier. It requires less time and users gain a better understanding of the overall system.

Since users began using AutoProducts (even with limited capabilities), we have received positive feedback about the ease-of use and efficiency of FDS.

Although AutoProducts required a significant effort to develop because of the complexity of the interfaces involved, its use will provide significant cost savings through reduced operator time and maximum product reliability. In addition, user satisfaction is significantly improved and flight dynamics experts have more time to perform valuable analysis work. AutoProducts helps both analysts and developers do their jobs more efficiently and effectively.

Glossary of Terms and Acronyms

curses	character based windowing library
EMOS	EOS Mission Operations System
EOS	Earth Observing System
FDIR	fault detection isolation and recovery
FDS	AM-1 Flight Dynamics System
FORTRAN	formula translation, a programming language
FTP	file transfer protocol
HP-UX	Hewlett Packard’s flavor of UNIX
MATLAB	MATrix LABoratory by The Math Works, Inc.
MFPU	Magnetic Field Prediction Utility
NASA	National Aeronautics and Space Administration
Perl	Practical Expression and Report Language
Perl/Tk	Perl Toolkit
STK	Satellite Tool Kit
TAM	Three-Axis Magnetometer
Tcl/Tk	Tool Command Language Toolkit
TDRS	Tracking and Data Relay Satellite
TGSS	TONS Ground Support System
TONS	TDRS On-board Navigation System
UNIX	a multi-user operating system
Windows NT	a Microsoft operating system